INT305
Semi-structured and Unstructured Data Management

Firebase Database Part 2

Semester 1/2023
Sanit Sirisawatvatana

Add data to Cloud Firestore

- There are several ways to write data to Cloud Firestore:
  - Set the data of a document in a collection
  - Add a new document to a collection. In this case, Cloud Firestore automatically generates the document identifier.
  - Create an empty document with an automatically generated identifier, and assign data to it later

https://firebase.google.com/docs/firestore/manage-data/add-data

Set a document

- To create or overwrite a single document, use the setDoc( ) method:

  setDoc(documentReference, dataObject) : Promise<void>

- Writes to the document referred to by the specified DocumentReference.

- If the document does not yet exist, it will be created.

- The dataObject is an object of data we want to store.

- When using the setDoc, you must specify an ID for the document to create.

Set a document

```
import { doc, setDoc } from "firebase/firestore";

// Add a new document in collection "cities"
await setDoc(doc(db, "cities", "LA"), {
  name: "Los Angeles",
  state: "CA",
  country: "USA"
});
```

doc(firestoreInstance, 'collectionName', 'documentID') : DocumentReference

- The document reference is a method that takes three arguments
  - The configure Firestore instance.
  - A string of the collection we want to store to. If the collection doesn't exist, it will be created automatically.
  - A unique document ID or name.

Set a document with merge option

```
import { doc, setDoc } from "firebase/firestore";

const cityRef = doc(db, 'cities', 'BJ');
await setDoc(cityRef, { capital: true }, { merge: true });
```

- You can provide merge or mergeFields, the provided data can be merged into an existing document.

setDoc(documentReference, dataObject, options) : Promise<void>

- If you're not sure whether the document exists, pass the option to merge the new data with any existing document to avoid overwriting entire documents.

https://firebase.google.com/docs/firestore/manage-data/add-data#set_a_document

Data types

```
import { doc, setDoc, Timestamp } from "firebase/firestore";

const docData = {
    stringExample: "Hello world!",
    booleanExample: true,
    numberExample: 3.14159265,
    dateExample: Timestamp.fromDate(new Date("December 10, 1815")),
    arrayExample: [5, true, "hello"],
    nullExample: null,
    objectExample: {
        a: 5,
        b: {
            nested: "foo"
        }
    }
};
await setDoc(doc(db, "data", "one"), docData);
```

Add a document

- When you use setDoc( ), you must specify an ID for the document to create.
- But sometime these isn't a meaningful ID for the document, and it's more convenient to let Cloud Firestore auto-generate an ID by calling addDoc( ).

addDoc(collectionReference, dataObject) : Promise<DocumentReference>
collection(firestoreInstance, 'collectionName') : CollectionReference

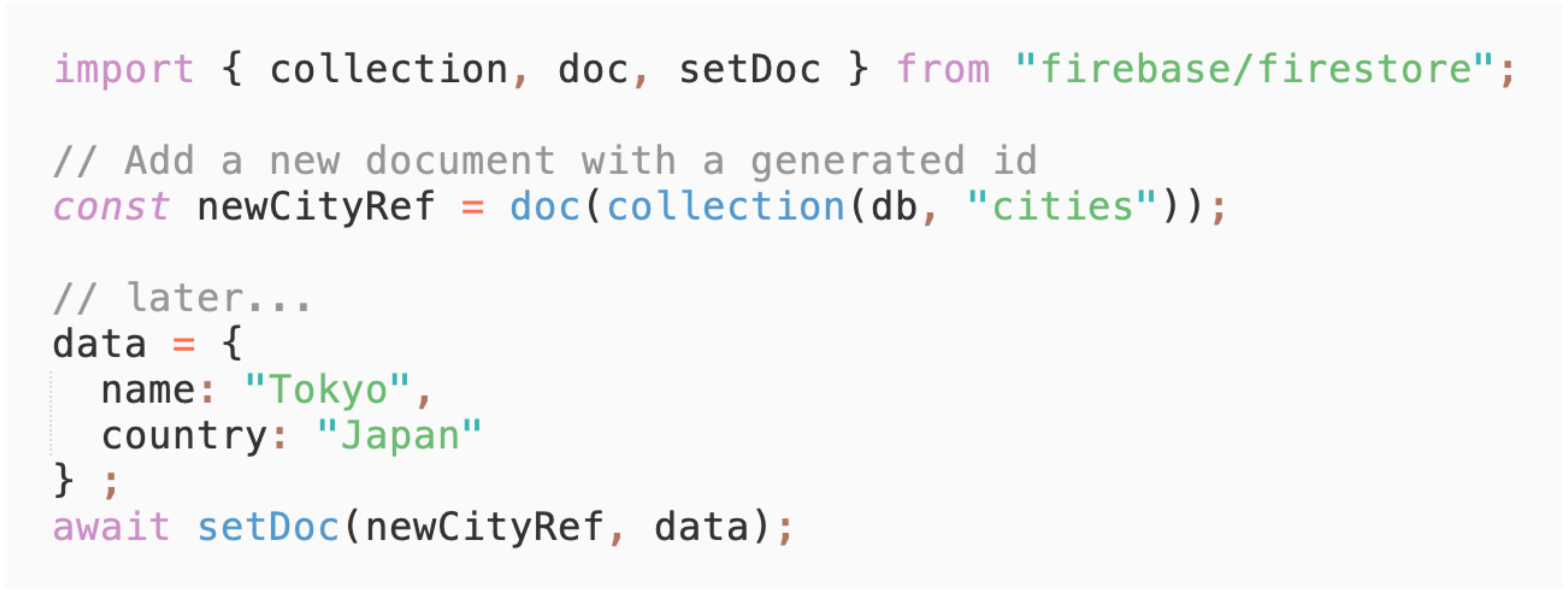
- Add a new document to specified CollectionReference with the given data, assigning it a document ID automatically.

```
import { collection, addDoc } from "firebase/firestore";

// Add a new document with a generated id.
const docRef = await addDoc(collection(db, "cities"), {
  name: "Tokyo",
  country: "Japan"
});
console.log("Document written with ID: ", docRef.id);
```

Create an empty document

- In some cases, it can be useful to create a document reference with an auto-generated ID, then use the reference later.

- For this use case, you can call doc( ):



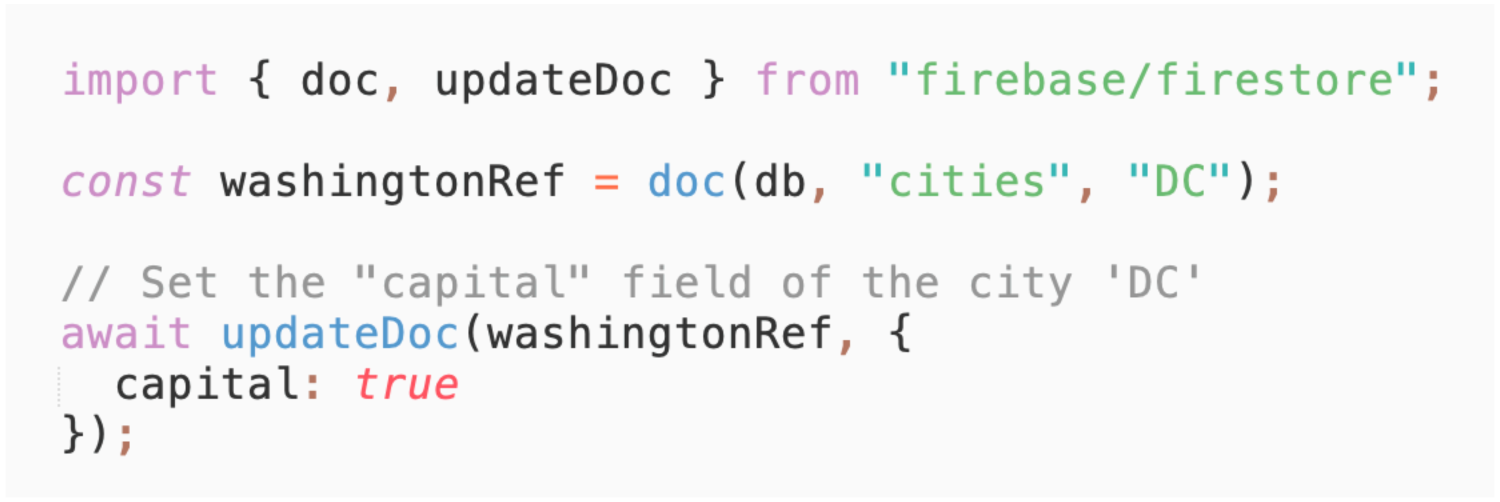
```
import { collection, doc, setDoc } from "firebase/firestore";

// Add a new document with a generated id
const newCityRef = doc(collection(db, "cities"));

// later...
data = {
  name: "Tokyo",
  country: "Japan"
} ;
await setDoc(newCityRef, data);
```

Update a document

- To update some fields of a document without overwriting the entire document, use the updateDoc( ) method.

  updateDoc(documentReference, dataObject) : Promise<void>

- The update will fail if applied to a document that does not exist.



```
import { doc, updateDoc } from "firebase/firestore";

const washingtonRef = doc(db, "cities", "DC");

// Set the "capital" field of the city 'DC'
await updateDoc(washingtonRef, {
  capital: true
});
```

Server Timestamp

- You can set a field in your document to a server timestamp which tracks when the server receives the update.

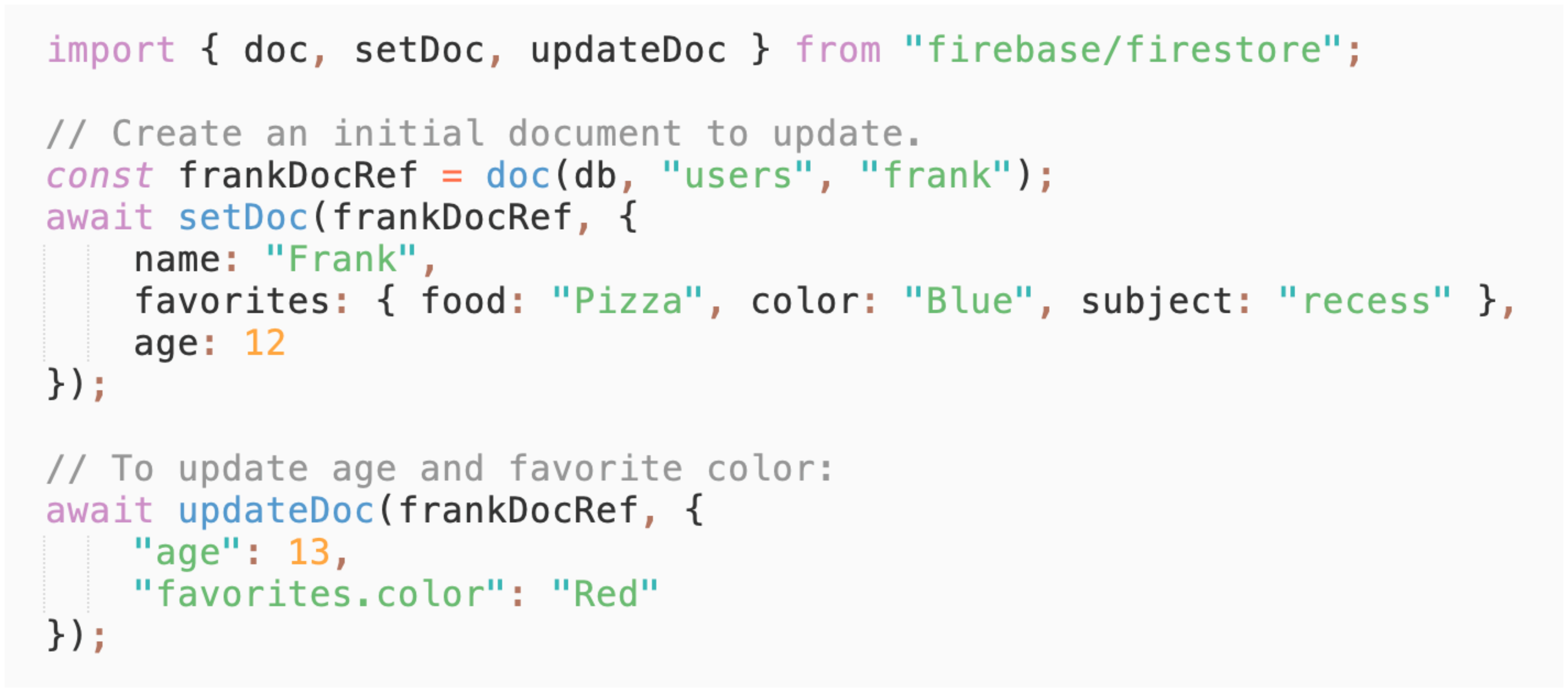
```
import { updateDoc, serverTimestamp } from "firebase/firestore";

const docRef = doc(db, 'objects', 'some-id');

// Update the timestamp field with the value from the server
const updateTimestamp = await updateDoc(docRef, {
    timestamp: serverTimestamp()
});
```

https://firebase.google.com/docs/reference/js/firestore_.md?authuser=0#servertimestamp

Update fields in nested objects

- If your document contains nested object, you can use "dot notation" to reference nested fields within the document when you call updateDoc( ).

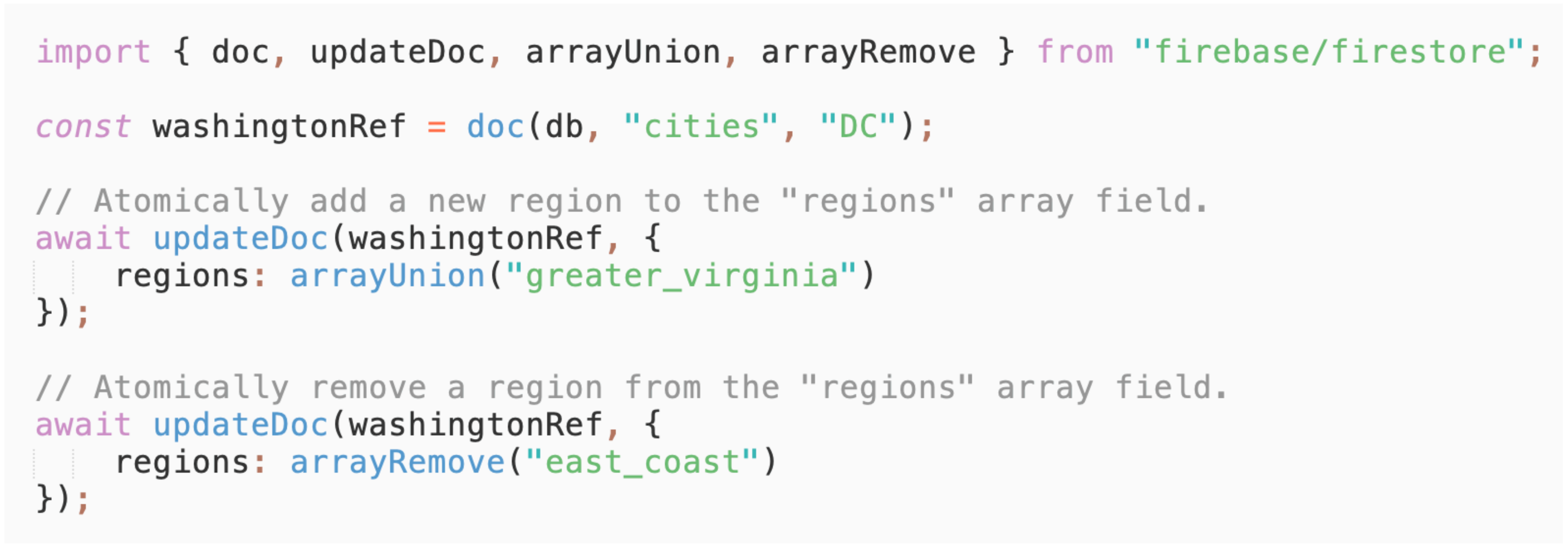
- Dot notation allows you to update a single nested field without overwriting other nested field.



```
import { doc, setDoc, updateDoc } from "firebase/firestore";

// Create an initial document to update.
const frankDocRef = doc(db, "users", "frank");
await setDoc(frankDocRef, {
    name: "Frank",
    favorites: { food: "Pizza", color: "Blue", subject: "recess" },
    age: 12
});

// To update age and favorite color:
await updateDoc(frankDocRef, {
    "age": 13,
    "favorites.color": "Red"
});
```

Update elements in an array

- If your document contains an array field, you can use arrayUnion( ) and arrayRemove( ) to add and remove element.
- arrayUnion( ) adds elements to an array but only elements not already present.
- arrayRemove( ) removes all instances of each given element.



```
import { doc, updateDoc, arrayUnion, arrayRemove } from "firebase/firestore";

const washingtonRef = doc(db, "cities", "DC");

// Atomically add a new region to the "regions" array field.
await updateDoc(washingtonRef, {
    regions: arrayUnion("greater_virginia")
});

// Atomically remove a region from the "regions" array field.
await updateDoc(washingtonRef, {
    regions: arrayRemove("east_coast")
});
```

Increment a numeric value

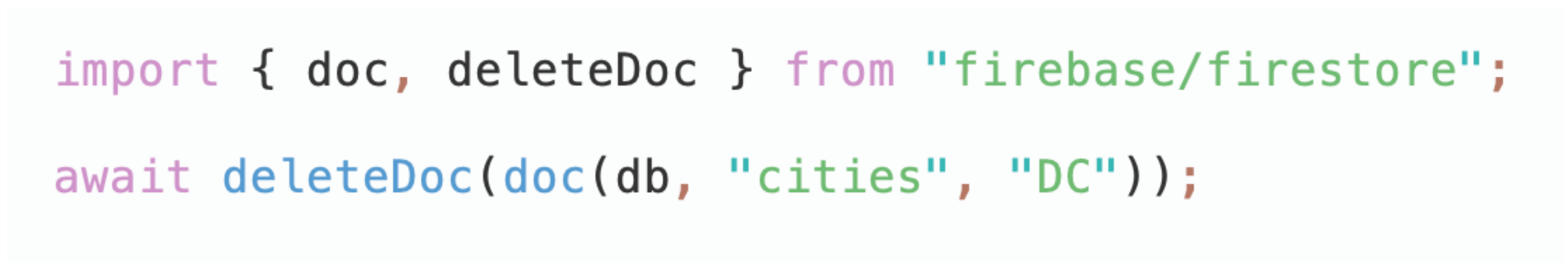
- You can increment or decrement a numeric field value.
- An increment operation increases or decreases the current value of a field by the given amount.
- If the field does not exist or if the current field value is no a numeric value, the operation sets the field to the given value.

```
import { doc, updateDoc, increment } from "firebase/firestore";

const washingtonRef = doc(db, "cities", "DC");

// Atomically increment the population of the city by 50.
await updateDoc(washingtonRef, {
    population: increment(50)
});
```

Delete data

- To delete a document, use the deleteDoc( ) method.

  deleteDoc(documentReference) : Promise<void>



```
import { doc, deleteDoc } from "firebase/firestore";

await deleteDoc(doc(db, "cities", "DC"));
```

- When you delete a document, Cloud Firestore does not automatically delete documents within its subcollections. You can still access the subcollection documents by reference.

- If you delete the ancestor document /mycoll/mydoc, you can still access the document path /mycoll/mydoc/mysubcoll/mysubdoc.

Delete fields

- You can delete individual fields from a document by specifying the deleteField( ) method in updateDoc.

  updateDoc(documentReference, { field: deleteField( ) })

```javascript
import { doc, updateDoc, deleteField } from "firebase/firestore";

const cityRef = doc(db, 'cities', 'BJ');

// Remove the 'capital' field from the document
await updateDoc(cityRef, {
    capital: deleteField()
});
```

Delete collections

- To delete an entire collection or subcollection in Cloud Firestore, retrieve all the documents within the collection or subcollection and delete them.
- If the collections are large, you may delete the documents in smaller batches to avoid out-of-memory errors. Repeat the process until you've deleted the entire collection or subcollection.
- Deleting a collection requires coordinating an unbounded number of individual delete requests. If you need to delete entire collection, do so only from a trusted server environment.
- Deleting collections from mobile/web client is not recommended according to performance and security reasons.

https://firebase.google.com/docs/firestore/manage-data/delete-data#collections

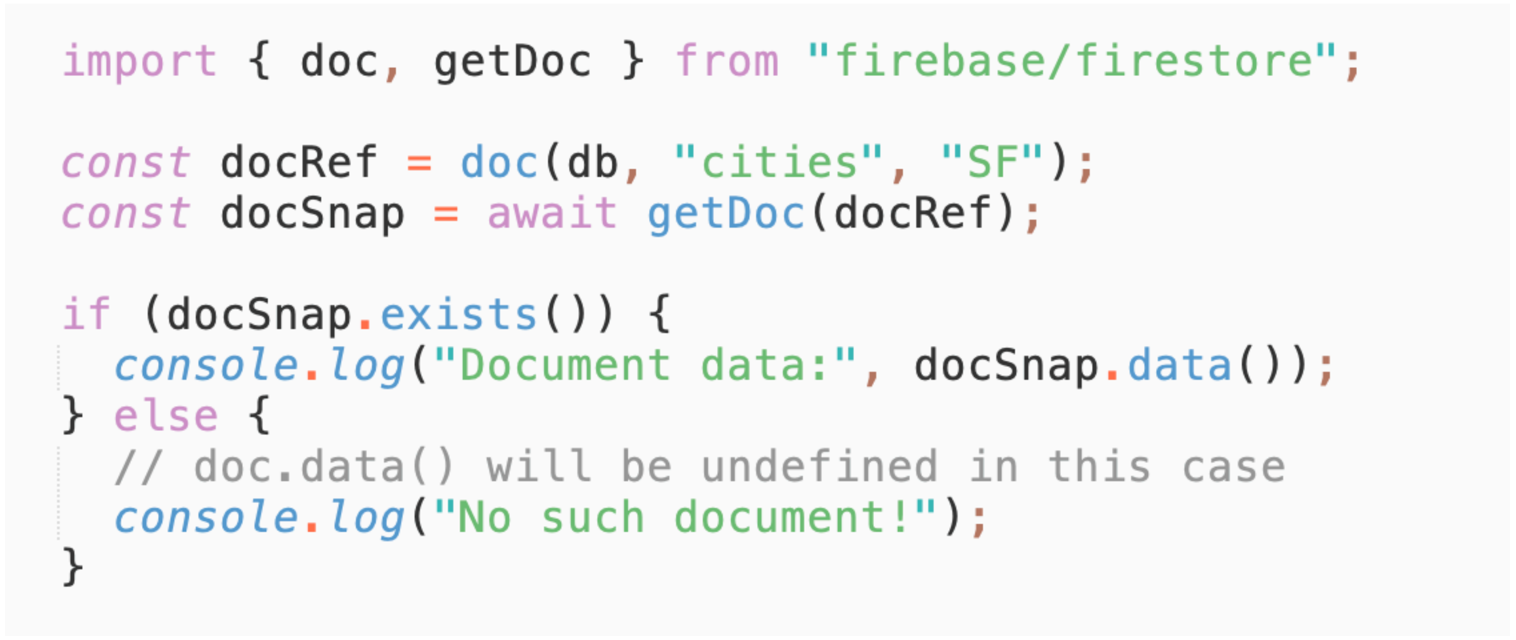
Get data with Cloud Firestore

- There are three ways to retrieve data stored in Cloud Firestore.
    - Call a method to get the data once.
    - Set a listener to receive data-change events.
    - Bulk-load Firestore snapshot data from an external source via data bundles.

https://firebase.google.com/docs/firestore/query-data/get-data

Get a document

- To retrieve the contents of a single document using getDoc( ) method.
  snapshot = getDoc(documentReference)
  snapshot.data( ) // full object
  snapshot.data( ).field // single field (key)
- Use exists( ) method to check if the document we're trying to fetch exists.



```
import { doc, getDoc } from "firebase/firestore";

const docRef = doc(db, "cities", "SF");
const docSnap = await getDoc(docRef);

if (docSnap.exists()) {
  console.log("Document data:", docSnap.data());
} else {
  // doc.data() will be undefined in this case
  console.log("No such document!");
}
```

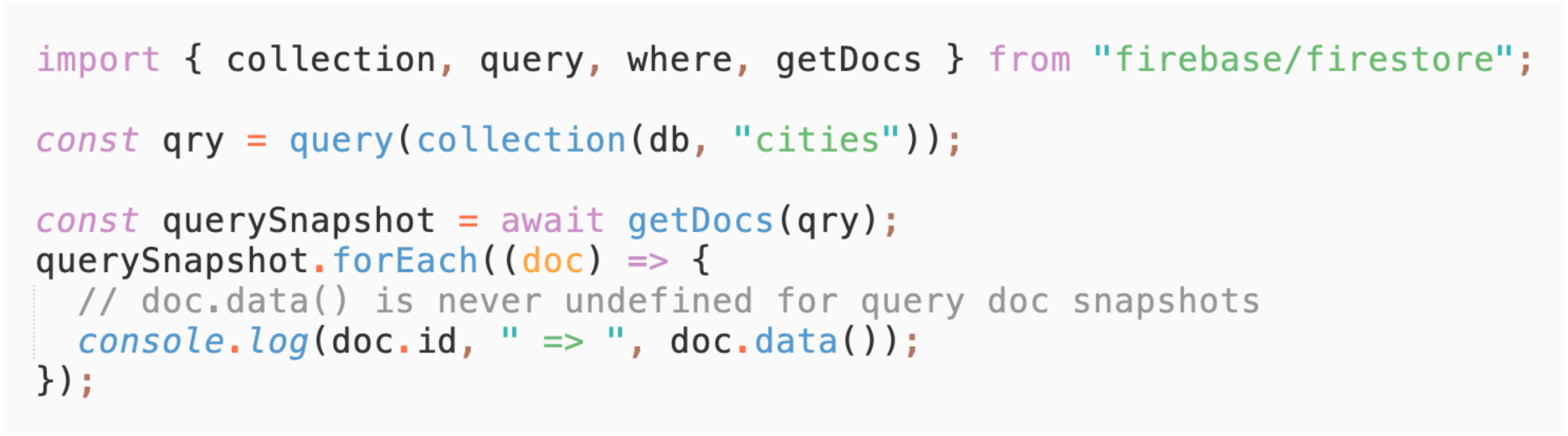
Get multiple documents from a collection

- You can get multiple documents with the getDocs( ) method.

  getDocs(query)

  query(collectionReference, queryConstraints) : Query

- By default, Cloud Firestore retrieves all documents that satisfy the query in ascending order by document ID, but you can order and limit the data returned.



```
import { collection, query, where, getDocs } from "firebase/firestore";

const qry = query(collection(db, "cities"));

const querySnapshot = await getDocs(qry);
querySnapshot.forEach((doc) => {
  // doc.data() is never undefined for query doc snapshots
  console.log(doc.id, " => ", doc.data());
});
```

https://firebase.google.com/docs/reference/js/firestore_.md?authuser=0#query

Query Constraint

- You can use where( ) to query all of the documents that meet a certain condition.

where(field, operator, value) : QueryConstraint

```javascript
import { collection, query, where, getDocs } from "firebase/firestore";

const qry = query(collection(db, "cities"), where("capital", "==", true));

const querySnapshot = await getDocs(qry);
querySnapshot.forEach((doc) => {
  // doc.data() is never undefined for query doc snapshots
  console.log(doc.id, " => ", doc.data());
});
```

https://firebase.google.com/docs/reference/js/firestore_.md?authuser=0#where

Get all documents in a collection

```
import { collection, getDocs } from "firebase/firestore";

const querySnapshot = await getDocs(collection(db, "cities"));
querySnapshot.forEach((doc) => {
  // doc.data() is never undefined for query doc snapshots
  console.log(doc.id, " => ", doc.data());
});
```

- In addition, you can retrieve all documents in a collection by omitting the where( ) filter entirely.

https://softauthor.com/firebase-firestore-get-documents-data-in-collection/

API Summary

API	Return	Description
doc(db, "colName", "objID")	Document Reference	Get Document Ref
doc(collRef)	Document Reference	Get Document Ref
collection(db, "colName")	Collection Reference	Get Collection Ref
setDoc(docRef, dataObj, options)	Promise<void>	Create/update a doucment
addDoc(collRef, dataObj)	Promise<Document Reference>	Add a new document
updateDoc(docRef, dataObj)	Promise<void>	Update the exist document
deleteDoc(docRef)	Promise<void>	Delete a document
getDoc(docRef)	Query Snapshot	Get a document
query(colRef, queryContraint)	Query	Get query
where(field, operator, value)	Query Constraint	Set query constraint
getDocs(query)	Array of Query Snapshot	Get documents

References

- https://firebase.google.com/docs/firestore
- https://www.koderhq.com/tutorial/vue/firestore-database/
- https://softauthor.com/add-firebase-to-javascript-web-app/
- https://googleapis.dev/nodejs/firestore/latest/Firestore_.html
- https://firebase.google.com/docs/reference/js/v8/firebase.firestore